Arto Sänkiaho

Realtime Hardware Monitoring

Helsinki Metropolia University of Applied Sciences

Bachelor of Engineering

Information Technology

Thesis

24 April 2014

Metropolia

Author(s) Title Number of Pages Date	Arto Sänkiaho Realtime Hardware Monitoring 41 pages 24 April, 2014
Degree	Bachelor of Engineering
Degree Programme	Information Technology
Specialisation option	Software Engineering
Instructor(s)	Erja Nikunen, Principal Lecturer Tomas Nordman, Section Manager

The goal of this thesis was to create a proof of concept tool for real-time quality monitoring called HAUDI. Its purpose is to provide an easy and consistent way for everyone in the organization to check and track the state of the hardware quality. This should be possible to do without thorough knowledge of the underlying technology.

HAUDI was developed using already existing software as a platform. There were two reasons for this; first by integrating HAUDI with already widely distributed software, HAUDI will be also widely distributed when they are integrated and released together as one package. The second reason is to use the services of the platform software. This makes the implementation simpler and reduces the amount of work and complexity of this proof of concept.

As a result, HAUDI was successfully integrated with the platform software and it brought quality related information easily available. Moreover it showed for the first time quality related data as a time series form which gave valuable information and helped to justify the need for a tool like HAUDI. Quality related data is now being successfully collected and stored for later analysis and issue tracking. The development of HAUDI led to interest towards it and the next step is productizing it and the release of it.

As a conclusion, it is clear that a tool such as HAUDI is needed. It has great potential and it can improve the quality control process to make the level of quality even better.

Keywords	Real-time, Hardware Monitoring, Proactive Quality, Web application, Data Mining.

Tekijä Otsikko Sivumäärä Aika	Arto Sänkiaho Realtime Hardware Monitoring 41 sivua 24.4 2014
Tutkinto	insinööri (AMK)
Koulutusohjelma	tietotekniikka
Suuntautumisvaihtoehto	ohjelmistotekniikka
Ohjaajat	Erja Nikunen, Yliopettaja Tomas Nordman, Section Manager

Tämän työn tarkoituksena oli tehdä prototyyppityökalu reaaliaikaiseen laadunvalvontaan. Työkalun nimi on HAUDI ja sen on tarkoitus tarjota helppokäyttöinen ja yhtenäinen tapa laadun seurantaan ja tarkkailuun koko organisaatiolle, ja tämän tulisi onnistua ilman syvällisiä tuntemusta seurattavasta järjestelmästä.

HAUDI kehitettiin käyttäen jo olemassa olevaa ohjelmistoa alustana ja näin tehtiin pääasiassa kahdesta syystä. Kun HAUDI integroidaan jo valmiiksi laajalle levitetyn ohjelmiston mukana, saadaan se tällätavoin helposti levitettyä sen mukana. Toinen syy tähän on alustana olevan ohjelmiston tarjoamat valmiit palvelut joiden käyttäminen helpottaa HAUDI:n suunnittelua ja toteutusta.

Työn tuloksena HAUDI saatiin onnistuneesti integroitua alustajärjestelmään. Syntyneen laatutiedon perusteella nähtiin ensi kertaa laatutietoa aikasarjoina joiden analysointi tuotti arvokasta tietoa ja laatutietoa kerätään ja tallennetaan tällä hetkellä HAUDI:n avulla lisää myöhempää analysointia varten. HAUDI:n kehitys johti kiinnostuksen lisääntymiseen sitä kohtaan ja seuraava askel onkin sen tuotteistaminen.

Johtopäätöksenä voidaan sanoa, että HAUDI:n tapaiselle työkalulle on tarvetta, ja sillä on suuri potentiaali laadunhallintaprosessin parantamisessa ja laadun parantamisessa entisestään.

Avainsanat	Real-time, Hardware Monitoring, Proactive Quality, Web application, Data Mining.

Contents

Metropolia

List of Abbreviations

ACID	Atomicity, Consistency, Isolation, Durability is the set of properties to ensure that database transactions are handled properly.
ATM	Asynchronous Transfer Mode
CPP	Connectivity Packet Platform.
D3	Data-Driven Documents
DBMS	Database Management System
HAUDI	Hardware Audit, the software tool described in this thesis.
HTML	HyperText Markup Language
HTTP	Hypertext Transfer Protocol, protocol used to transfer data in worldwide web
IMS	IP Multimedia subsystem
IP	Internet Protocol
ITK	ISP Tool Kit
JSON	JavaScript Object Notation
M-MGw	Media Gateway for Mobile Networks.
MSS	Mobile Soft switch Solution
RPC	Remote Procedure Call, a form of inter process communication.
SVG	Scalable Vector Graphics
URL	Uniform Resource Locator

1 Introduction

Ericsson is a world-leading manufacturer of telecommunication equipment and services. More than 40 percent of the world's mobile traffic passes through Ericsson's networks and there are over 1000 networks provided by Ericsson.

The goal of this thesis is to create a proof of concept, a software tool for the proactive real-time hardware quality monitoring. At first the tool will be focusing on the M-MGw nodes and later the scope will be expanded to cover other node types to unleash its full potential.

The need to create this tool originates from the fact that information about hardware issues does not travel between organizations well enough. Improving this information flow would help organizations to co-operate better together. There is also a clear need for a consistent way for individuals from different organizations to track and search for hardware issues and their root causes. Moreover the information about the hardware issues is not being systematically collected from the live nodes today. A tool which satisfies these needs will bring significant value for its users and organizations. In addition the tool will be made available for everyone, hence it must be easy to use and intuitive.

The tool created is called HAUDI which is an abbreviation of Hardware Audit. HAUDI is a tool which will systematically and regularly collect and store information from hardware. The information is stored so it is possible to create statistics out of the data. This statistics can reveal causes for hardware issues that would be otherwise hard to find. In addition cross referencing data and statistics can reveal connections between issues which would be almost impossible to find otherwise. Other possible use cases for HAUDI are for instance quality benchmarking and as a tool to provide help for customer support.

The following chapters deal with the environment in which HAUDI operates, hardware quality, existing studies and tools and the design and implementation of HAUDI.

2 Hardware Quality

To be able to design and implement a quality monitoring tool it's important to understand what quality is and why quality is important.

This chapter deals with the following topics:

- What is quality ?
- Why is quality important?
- What benefits and opportunities does quality monitoring bring?

2.1 What is quality?

Quality is value for someone. There are two definitions for quality objective and subjective.

The definition of subjective quality is a customer-based definition of quality. This simply means that the customer experiences the product or service well done and good. This sounds simple but it is not. What represents quality to you might not be quality for someone else. Quality is something everyone has an opinion about. Moreover the opinions of others tend to have an effect on the person’s opinion about quality. This can lead to a situation where people have a sense of bad quality of something that they haven’t even used. If a product has this kind of reputation it can be really hard to discard it later on. In addition the criteria of subjective quality tend to change over time and what is quality today might not be considered quality anymore after ten years [1 p. 10]. Subjective quality can be followed and measured, but it’s not as precise and easy as in the case of objective quality.

The definition of objective quality is the statistical definition of quality. This is a more scientific and precise approach to define quality based on mathematics. To put it simply, this means looking for variation between what the customers ask and what you produce, with less variation meaning better quality. Products or services always have so called natural variation and it is the abnormalities that are searched for by using statis-

tical methods. [1, p. 11] The good thing about objective quality is the fact that it can be measured easily because it's based on mathematics. In addition its improvement can be followed thanks to this.

2.2 Importance of hardware quality

By improving quality two important goals can be achieved. First the customer satisfaction improves, and second, the number of defects reduces. Clearly both of these matters are important as they can greatly reduce costs and help maintain good relations with customers. [2, p. 3].

Hardware quality is equally important as software quality. Hardware problems are sometimes hard to detect in factory and integration tests. The main reason for this is that problems can be caused by different hardware and software setup combinations, possible environmental matters, different system load levels etc. Needless to say, testing all the different possible hardware setup combinations is virtually impossible. Moreover the repairing and changing of the defected hardware is expensive and sometimes difficult when hardware is already in live use.

The impact on the system caused by the hardware defect is usually devastating. This makes it crucial to find hardware issues as soon as possible, or even better, before they occur on the field. Therefore, proactivity is important in quality and the best scenario would be the prediction of the hardware problems. This can only be achieved by proactively looking for the symptoms of hardware issues.

2.3 Benefits of real time hardware monitoring

The benefits of real-time quality monitoring are great. Possibly the greatest benefit is that it makes possible to proactively search for symptoms of possible upcoming issues. In addition old and new issues can be cross-referenced with data, for example with software level or firmware version. This type of data analysis can reveal a lot of valuable information. The information is valuable not only for hardware designers but also for example for software designers, testers and quality engineers. Above all the hardware quality information should be available for everyone who might benefit from it.

With real-time quality monitoring it's possible to find causes for issues that depend on many different factors. With trend analysis and cross-referencing the data an individual might be able to spot a cause for the issue that would otherwise be virtually impossible to see or find without luck.

Without real-time monitoring, a cause for some hardware issue can be misinterpreted. In the worst case this leads to replacement of defected hardware and the new hardware can be defected again because the real issue was not fixed. To put it in another way, misinterpreting the issue can cause replacement of non-defected hardware because the issue was elsewhere. These types of problems can be avoided with real-time monitoring. It could be easy to see, for instance, that certain problems started only after certain new software version was introduced and this can almost certainly rule out hardware issues.

3 Existing studies and tools

This chapter takes a closer look at the existing hardware quality monitoring tools and studies.

3.1 Tools

The hardware quality monitoring tools can be sorted in two groups.

- The tools which log the data
- The tools which log and/or process & display the data.

The tools which log only data are common in computers for instance. Most of the Unix like operating systems collect data with syslog daemon which implements the syslog protocol. In addition to computers, the syslog protocol is also used in printers and routers for instance and it is a standardized way of logging data. [3]

The tools which log and/or process & display the data consist of wide variety of tools for different purposes. There are lots of commercial and noncommercial tools for differ-

ent purposes. Most of the tool focus on server and network performance and fault monitoring.

As an example of the noncommercial tools Nagios is a tool to monitor computer systems and networking. Nagios has alerting and monitoring capabilities and it works with servers, switches, applications and services. Nagios has system resource monitoring support (disk, cpu etc.) it can also monitor e.g. temperatures and voltage levels. Nagios offers web interface where the user can check the reports of current status of the monitored system. The interface can be extended with plugins that can for instance draw charts out of desired data. [4]

Another example of the noncommercial tools is Munin. Munin is a computer system monitoring tool much like Nagios. Munin remembers what it finds and it presents all the information in charts. [5]

In addition to these there are tools to monitor proprietary systems, for instance cable TV network, phone call center and power supply monitoring software just to mention few.

3.2 Studies

Hardware quality monitoring is closely related to fault detection and isolation which is studied a great deal in medical and aerospace industries. Fault detection and isolation consist of monitoring the system, identifying when faults occur and finding out the type and location of the fault. Fault detection and isolation is part of and studied by control theory which is a branch of engineering and mathematics. It deals with behavior of dynamic systems. [6] [7]

3.3 How these relate to HAUDI

HAUDI definitely falls in the scope of fault detection and isolation. It monitors the systems and identifies faults occurring in the system and finds out their type and location. This is exactly how fault detection and isolation was described before. Compared to the existing tools, HAUDI is somewhat different. HAUDI does not log the data, it only fetches it from the logger but HAUDI does process and display the data. The main differ-

ence compared to the majority of the tools is that HAUDI creates statistics out of the data, as opposed to simply drawing diagrams and graphs out of available data. This means that it does quite heavy processing of the stored data before it is displayed, which is something that most of the other tools do not necessarily do. Instead they just store the already processed data. Processing of the data on demand is mandatory because if only the processed data would be stored, it would not be possible to later process the original data anymore with different parameters or criteria and information is lost. Moreover, storing the processed and original data would be too much disk consuming and impractical.

4 HAUDI

This chapter deals with the description, requirements and limitations of HAUDI.

4.1 What is HAUDI?

HAUDI is a tool for proactive hardware quality monitoring. Its purpose is to proactively reveal issues that threaten hardware quality. HAUDI is a tool which should be a part of process and it should be used to improve quality. One way to do this is to create a channel for individuals from all over the organization to see what the current state of the hardware quality is. By doing this people will pay more attention to quality and begin to question why the hardware shows these indicators of quality problems. This will raise a question asking whether they really are hardware quality issues or false alarms. This forces people to take action and identify the reasons behind these issues and thus prevents them from happening again.

HAUDI will also clarify the source of the issues. Since HAUDI specifically scans for hardware quality issues one of its purposes is to make sure which issues actually are hardware related and which are not. Before HAUDI there hasn’t been any tool that scans specifically for hardware issues. The problem in this rises when there are problems in quality that might be hardware related but it’s impossible to say for sure without further and thorough examination. Figure 1 depicts the situation.

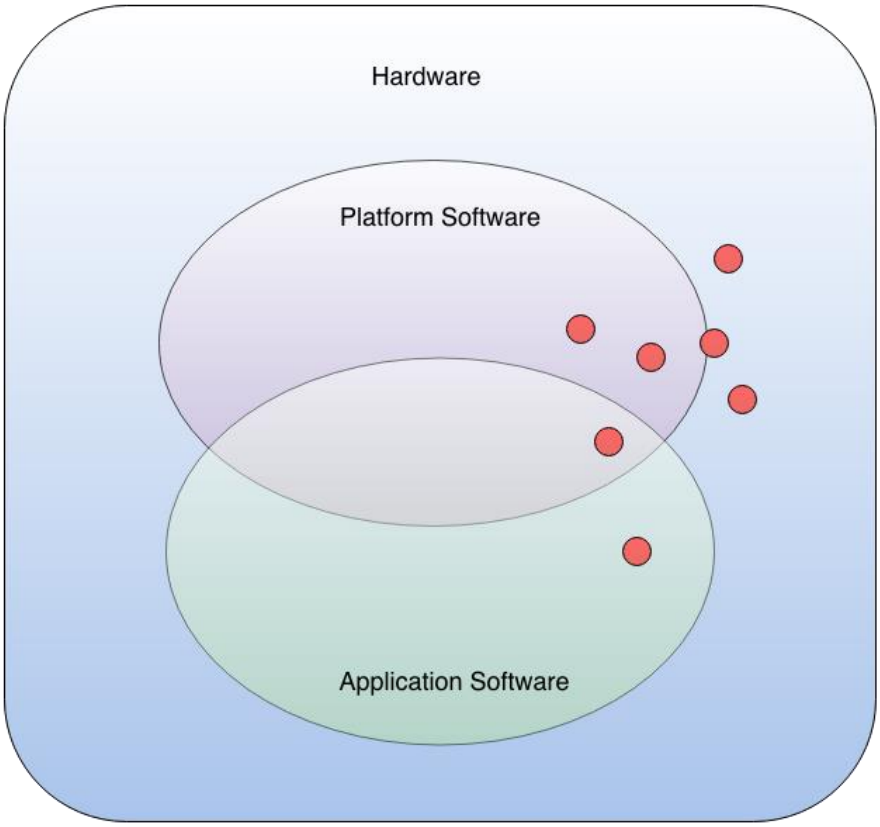

Figure 1. Errors in the node.

Figure 1 above shows the situation when there are errors (red dots) detected in node. The problem comes from the way how the software is layered on the hardware. First there is the platform software running. A platform software can be thought of as an operating system on which the application software runs. Now when there is an error that is seen in the application software, is the cause of the error in the application software itself or in platform software? Or is the cause hardware related? Moreover if there is a platform software error, is the cause in the platform software itself or in the underlying hardware? One of HAUDI's tasks is to distinguish the real hardware related issues from the software related issues.

Another issue that can be solved with HAUDI is the replacement of non-faulty boards. In other words, when a board is malfunctioning and the cause is hardware related, the board is replaced and the replaced board is sent for further examination. However it's not uncommon that when a faulty board is examined, there is no fault found at all and the board is functioning like it should. It is virtually impossible to see what caused this issue in some cases and this is what HAUDI also is designed to solve. By regularly scanning the node and its boards it is possible to see the symptoms which ultimately lead to malfunction. The reason for this comes from the fact that some of the error logs

in hardware are wiped out when the board is powered down. This effectively destroys any evidence that may help to figure out the cause of the problem. By regularly reading these volatile logs and by storing them, HAUDI makes it possible to investigate the causes of these failures later.

Why would someone use HAUDI? There are many reasons for this. First, HAUDI provides a consistent way to follow-up the quality of the hardware in real time and this information is valuable for hardware designers. This is because HAUDI provides means to follow up the quality of new hardware in real-time and this is a significant improvement to the current state of hardware monitoring. Moreover the people who are in charge of customer satisfaction are interested in the real-time status of quality. In addition HAUDI offers a way to measure the quality and this means that when the quality decreases or increases it's easy to see it immediately from HAUDI. As a conclusion, anyone who is interested, in charge or somehow related to the state of the current hardware quality will greatly benefit by using HAUDI.

In short HAUDI can be described as a tool that collects, processes and stores the data from hardware for statistical and quality purposes. HAUDI will be built as a web based software which means that it will be available for everyone who has access to the intranet. In addition, there will be no user registration required because that alone would reduce the user amount. Moreover, usability is an important factor from the user satisfaction point of view and it will be discussed in more detail later.

4.2 Workflow

The workflow of HAUDI is described next to shed some light on the functionality and to give a better picture of what it does.

First HAUDI collects the data from hardware logs. It has a schedule when data is fetched from different nodes and the default schedule is to get new logs once a day. When logs are being fetched, HAUDI issues specific low level hardware log commands towards the nodes and then archives and fetches the logs. At this point the archived logs are transferred to the HAUDI server.

Now it's time to process the data. First HAUDI extracts the archives and then it begins to parse the files. File parsing is a quick process, even if the log files are big. This is because slow database operations are not done at this point yet and all the work is done in the physical memory of the computer. At the same time when files are parsed, the parser inserts its findings in the logical structure of the node. The logical structure of the node is a tree like structure in memory which represents the hardware that the node has in it. In addition, the possible events, which are e.g. errors and alarms are also inserted in the logical structure.

At this point begins the most time consuming part of the process. HAUDI checks all its findings against the database and sees if those are already stored. The reason for this is the fact that HAUDI does not store redundant data. Another reason for this is the length of the log files. They can be several millions of rows long and there will be a large amount of database queries which causes this step to take a long time.

Finally the data is ready to be queried from the database through the web interface of HAUDI. The web interface of course will be still processing the data further to suit its needs. Figure 2 below shows an example how web interface displays the processed data.

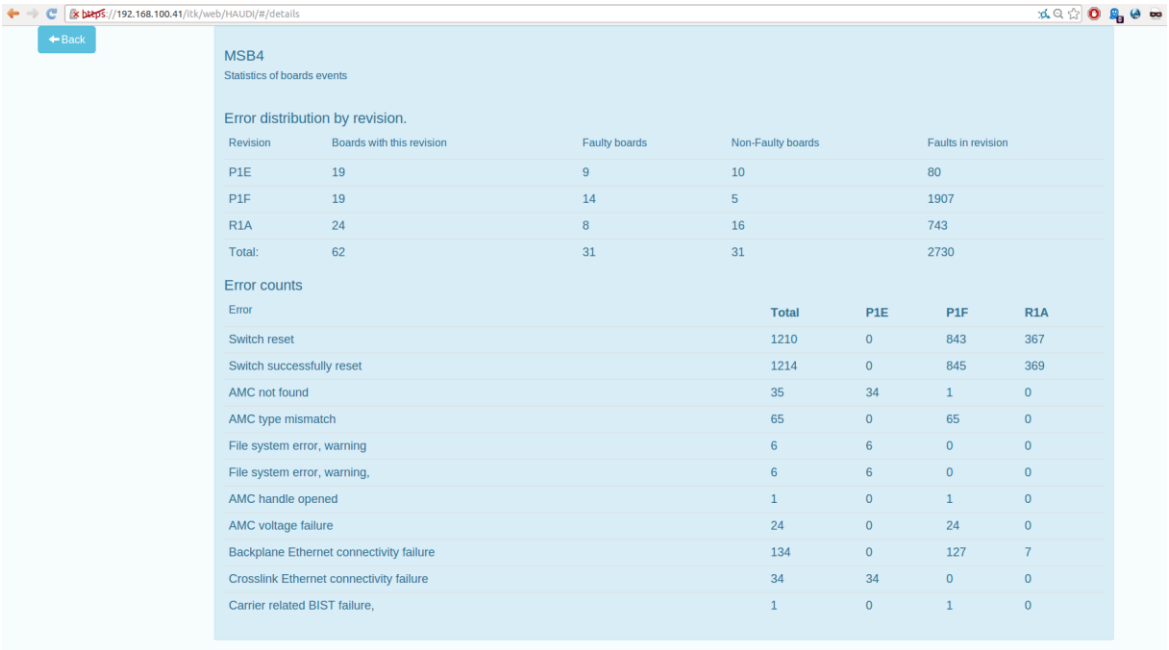

Figure 2. Web interface showing processed data.

The steps of workflow are more thoroughly described in a later chapter along with the implementation of HAUDI.

4.3 Requirements

HAUDI is designed to meet the following list of requirements.

1. To collect up-to date quality information from the hardware and create real-time statistic out of the quality data.

2. To create a tool which everyone in the organization can use without thorough knowledge of the underlying hardware.

3. To create a tool which everyone in the organization has access to.

4. To create a tool which offers an overview of the quality status instantly. The information should be available without any user input.

5. HAUDI must be integrated with ISP Tool Kit (ITK).

The list describes the most important points that are considered when HAUDI is designed and implemented. The points on the list are simple. Still, it is important to explain what the points really mean from the thesis point of view.

The first item on the list means that there must be a database which stores the data. In addition the data which is stored must be regularly fetched from the nodes. This is mandatory for keeping the data up-to-date. However, the data must be processed before it can be stored. The data processing consists of discarding unnecessary data, formatting the data correctly, checking that there is no redundant data and so on. There are some downsides to this. The data processing is resource intensive and will cause the system to slow down when the process is running. Moreover the regular data fetching can lead to a situation where there is too much data. This can cause the application to slow down, choke or in the worst case to crash.

The second item on the list is a strongly dependent on one important thing: the user experience. The user should get a clear understanding of the current status of the hardware quality. This means that the tool should be intuitive. It also should be self-explaining and offer something of value for its user, every error shown should have an

explanation and it should be clear what the errors mean. In general, everything the user sees must have some purpose so it needs to be carefully thought out. Moreover the user should be able to find the key information without problems.

The third item on the list means that the tool must be web-based. A web-based tool is available for everyone who has access to the intranet. In addition, the web application works on every operating system and with different devices. It is easy to keep updated because all it takes is to update the web application on the server. However there are problems too with the web applications. Internet browsers, especially the older ones have compatibility issues with new web technologies. It cannot be assumed that every user will have the latest version of a web browser installed. This means that technologies must be chosen carefully and some compromises must be done.

The fourth item on the list is also dependent on the user experience. It is virtually impossible to achieve instantly available information when the amount of data grows large. Luckily instant in this case means a time which is less than a second which is possible to achieve with smart design of the application and when the amount of data is reasonable. No matter what, at some point the amount of data grows too big and the one second time passes. At this point the loading time begins to feel long and loading indicator must be shown for the user. Another magical time limit is 10 seconds after which users will probably leave the web page. To make sure that loading times, even with large amounts of data stay as short as possible, it is mandatory that good performance is one of the design rules. Performance must be taken care of throughout the whole development process of HAUDI. [8]

The fifth item on the list means that HAUDI must use ITK`s interfaces and services. ITK is an abbreviation of ISP Tool Kit where ISP is an abbreviation of In Service Performance. ITK is an internal Ericsson tool to monitor node performance among other things. Because the goal is ultimately to release HAUDI with ITK it's important to respect and use as much as possible the same services, interfaces and conventions which ITK uses. This is to ease the integration and release process.

4.4 Limitations

Since HAUDI is a proof of concept there are certain constraints to limit the scope. HAUDI will be working only with CPP platform at the beginning. CPP is an abbreviation of Connectivity Packet Platform which is used as a platform in many different Ericsson products. Furthermore M-MGw nodes will be the main target at start. The relevant part of the network where M-MGw nodes are in is described next.

To see the big picture and where HAUDI fits in, the network and Ericsson Media Gateway for Mobile Networks (M-MGw) is covered first. From HAUDI point of view the M-MGw node is the most important because HAUDI will be using it as a source of quality data collection.

4.4.1 Mobile Softswitch Solution

The M-MGw is part of Mobile Soft switch Solution (MSS) core network. The MSS consists of two main elements, the M-MGw and Mobile Switching Center Server (MSC-S) nodes. MSS is a layered architecture where control and switching are separated to different nodes. The M-MGw node handles switching and the MSC-S handles control and they are located in connectivity and control layer respectively.

The M-MGw node is part of the layered network architecture. The layered architecture separates control and transport from each other. The Figure 3 below depicts the layered network structure [9, p. 4].

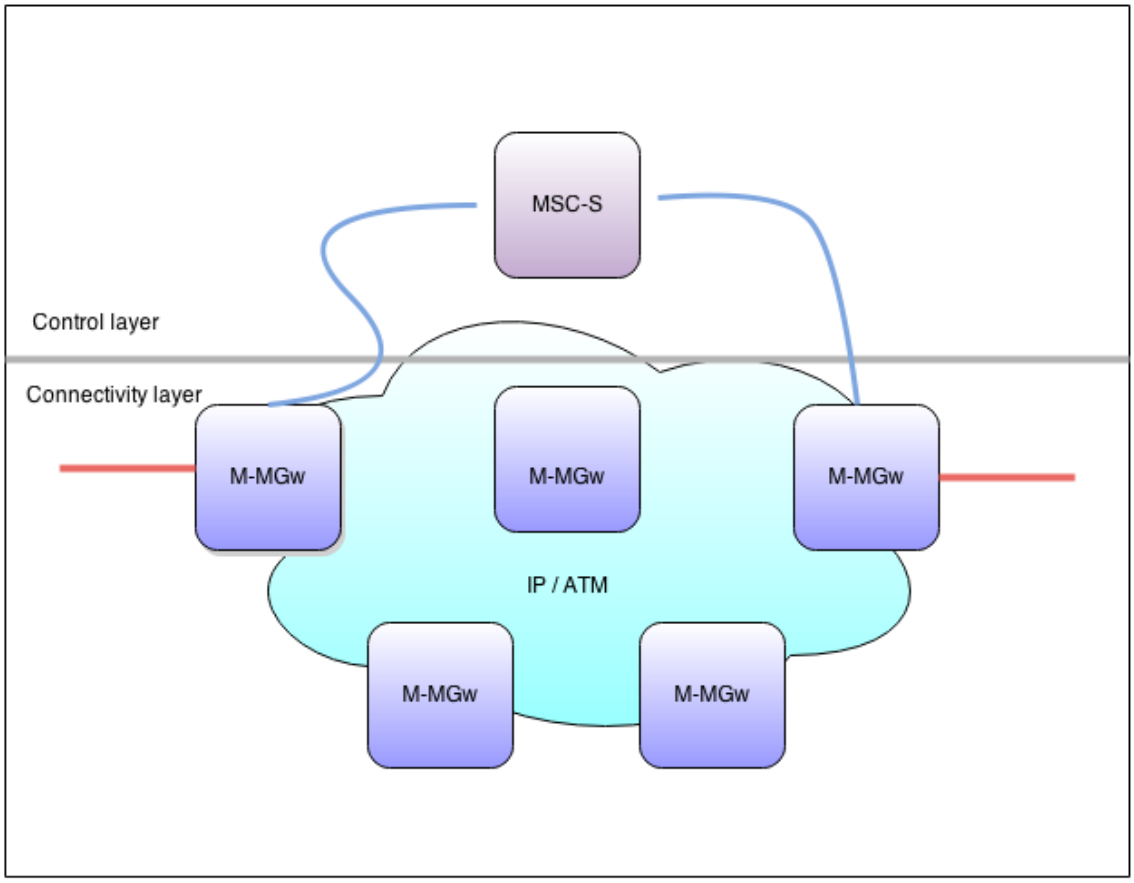

Figure 3. The layered network structure.

The MSC-S node is located in the control layer and it handles the signaling (blue lines) of the traffic in the network. The M-MGw nodes are on the connectivity layer and they are responsible for the actual payload (red lines) handling [9, p. 4].

The control layer controls the traffic and it makes sure that the services are high-quality and there are no interruptions. The control layer is not in the scope of this thesis and it is not discussed further.

The connectivity layer offers interface for different networks, for example 2G and 3G radio access networks. The connectivity layer handles the transport of the voice and data services. The M-MGw is a key component on the connectivity layer and it handles call switching by connecting calls to the correct network. In addition M-MGw provides protocol conversion, speech stream decoding and encoding among other services [9, p. 6].

4.4.2 Connectivity Packet Platform

CPP is a scalable hardware platform for the nodes which process the load of the mobile network. The M-MGw node is built on top of it but it is not the only node using the CPP platform. The CPP based node consists of racks in which the boards are installed. The number of boards varies based on what the function of the node is. The boards installed in the racks can be e.g. power supplying board or application device board. One rack can support up to 28 boards and one node can have multiple racks depending on what the configuration of the node is [10].

4.4.3 Ericsson Media Gateway for Mobile Networks, M-MGw

M-MGw connects the different networks and it enables different data transport technologies to be used together. The M-MGw connects the core network to e.g. radio access networks, IMS and VoIP networks. M-MGw also makes it possible to connect packet-switched telephone-network to circuit-switched telephone-network. In addition, M-MGw can do speech processing. Depending on the network topology, there can be one or more M-MGw nodes. This means that one node can handle the whole network or if necessary multiple nodes can be configured so that each node has a specific task. As a result of this load balancing, the network is easily scalable [10] [11].

5 Development of HAUDI

This chapter deals with the design and implementation of HAUDI. This is done in top-down manner, first looking at the overview and then describing each component in a more detailed way. Technologies that are used are introduced in the beginning of each section including the motivations for choosing them.

5.1 Overview

This section describes how HAUDI is built and designed. It describes each piece in detail using a top-down approach. HAUDI can be split into three distinct modules, back end, front end and the database. Furthermore, the back end can be split in two parts, the part which interacts with the database, and the other part which interacts with the front end. The front end can also be dissected into three smaller pieces. It consists of the data display component, services component and the views component. The architecture type of HAUDI is so called thick client model in which the front end does most of the work and asks for more data from back end only when it is necessary. There are several benefits in this design most notably following [12].

- Lower server requirements, because the client does most of the processing needed.
- The data visualization is handled at the client side. This makes it possible to create interactive and dynamic visualizations.
- Better server capacity. Server does less work because client processes the data and server can support more simultaneous clients.

Figure 4 below depicts the HAUDI overview structure.

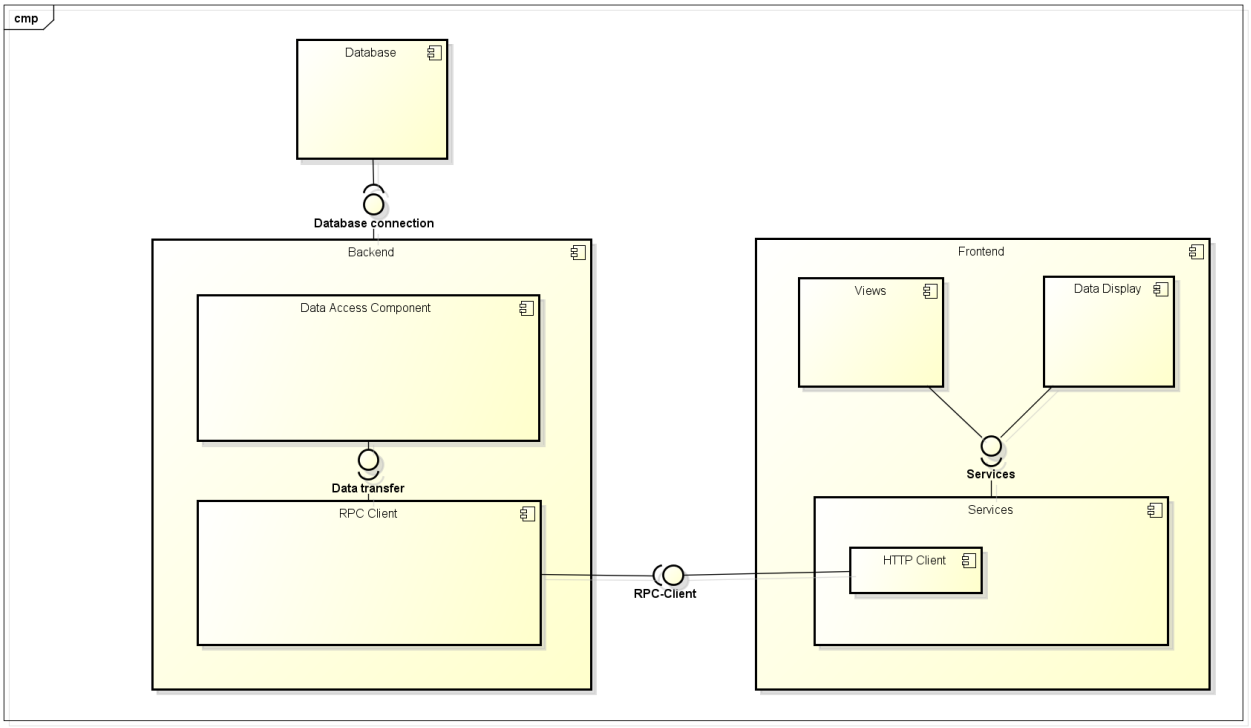

Figure 4. Overview of the HAUDI structure.

As Figure 4 shows, the backend consists of two major components, the RPC Client and the data access component. The purpose of RPC client is to route and handle parameters of data queries correctly for data access component. The RPC client is simple and it makes heavy use of ITK's services. The data access component on the other hand is more complex. It consists of the actual code which executes the database queries and it has the parser component which parses the data. The parser component also has the responsibility of inserting the data into the database correctly. The front end component can be split up in three major components. The first one is the views component which is the component the user will actually see. The second component is the data display component which is used with views together to display the data. The third component is the services component which has the data processing logic built in it. All of these components will be covered in more detail later in this chapter.

5.2 Usability

The usability of the tool is one of the key factors to its success. User experience is extremely important, because if the tool is impractical, the users don't want to use it.
The most important thing is the first impression. Users make permanent judgments about websites within a fraction of a second. If this impression is bad, it is hard to change it later.[13].

To make a good first impression the tool must be designed with a few simple rules of thumb.

The first and most important of these rules is, don't make the user think. What this means is that the website should be so simple and self-explanatory that the user won't have to think what something on the page is. It should be clear and intuitive enough for the user to immediately see and find what he is looking for without giving it a thought.

The second important rule of thumb in usability is the response time of the website. There are 3 different time limits that are important [14].

- 0.1 second is approximately the limit when user feels that the response is instant and user has the feeling of directly operating on the data.
- 1.0 Second is approximately the limit how long user's focus will remain uninterrupted. While the delay stays between 0.1 and 1.0 second no special feedback is needed but the feeling of direct operation of the data is lost.
- 10 seconds is the limit how long user will keep the attention focused. This means that displaying load dialogue will be enough for wait times lasting less than 10 seconds. Everything above this time should have indicator for remaining wait time so the user knows what to expect.

The third important rule of thumb is to make the website not to contain too much text. This is because people tend to spend little time reading web pages. Too much information in the form of text on a single page will overwhelm the user and this results in bad user experience [15 p. 22].

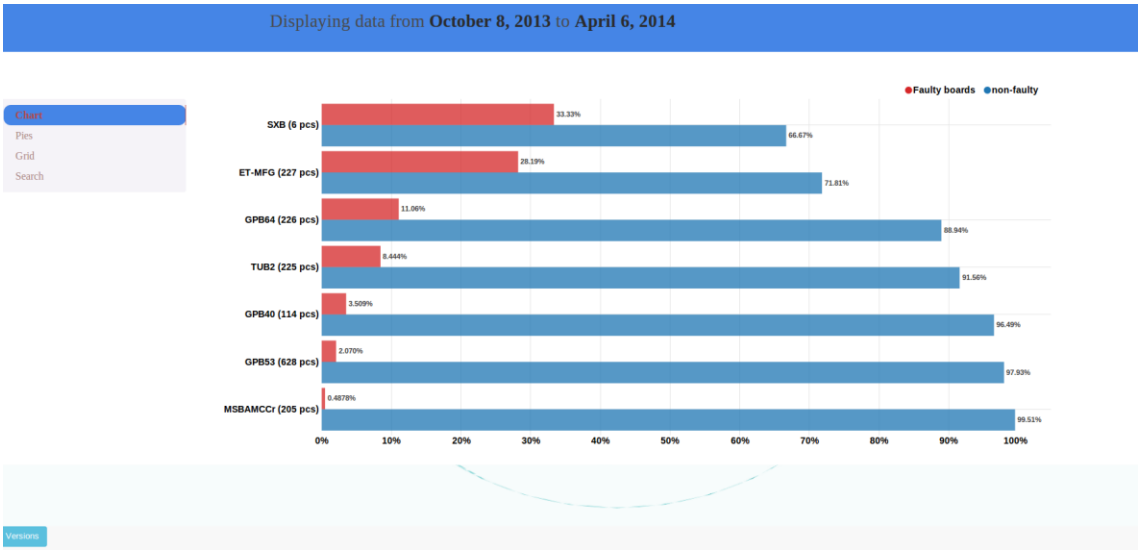

Figure 5. HAUDI overview is kept as simple as possible.

Obviously this is not a complete list of rules regarding how to design a website which has good usability. The rules presented are the ones which are the most relevant from the perspective of HAUDI.

5.3 Database

The database structure in a tool like HAUDI is extremely important. The database is the core of the application and poorly designed database structure can have a huge impact on the performance of the application. For instance if the relations of the tables are badly designed it can result in the queries becoming too complex and this can cause poor overall database query performance. In addition the amount of data collected by HAUDI will rapidly grow when the amount of the nodes where the data is collected from increases. Complex queries and big amount of data together will efficiently choke the whole application. Moreover, the database must be designed in such a way, that it is as simple as possible to allow adding more tables in it later if necessary.

The first thing covered is the database management system. After this the performance improvement techniques are discussed and at the end of the chapter, the database structure is described in detail.

5.3.1 Database management system

The database management system, DBMS hereinafter, which is used in HAUDI is PostgreSQL. PostgreSQL is an object-relational database management system. An object-relational database management system has support for an object-oriented database model. This means that it supports objects, classes and inheritance in database schemas. Moreover custom data types and functions are supported. The reason for this is to help and make it possible for object oriented programming languages to map the object to the database [16] [17].

PostgreSQL was chosen because ITK already uses it. Hence it is already available on the server where HAUDI will be deployed and so it is a natural choice. In addition PostgreSQL is robust and fully ACID compliant. It has support for important features to improve performance, such as database partitioning and replication. These features are necessary, because the amount of data stored in the database can grow rapidly and therefore, it is important to have DBMS which has good scalability [18].

5.3.2 Database structure overview

An overview of the database is covered first to help understand the structure of the database better.

The database structure of HAUDI consists of 18 tables, Figure 6 below shows the overview of the database components.

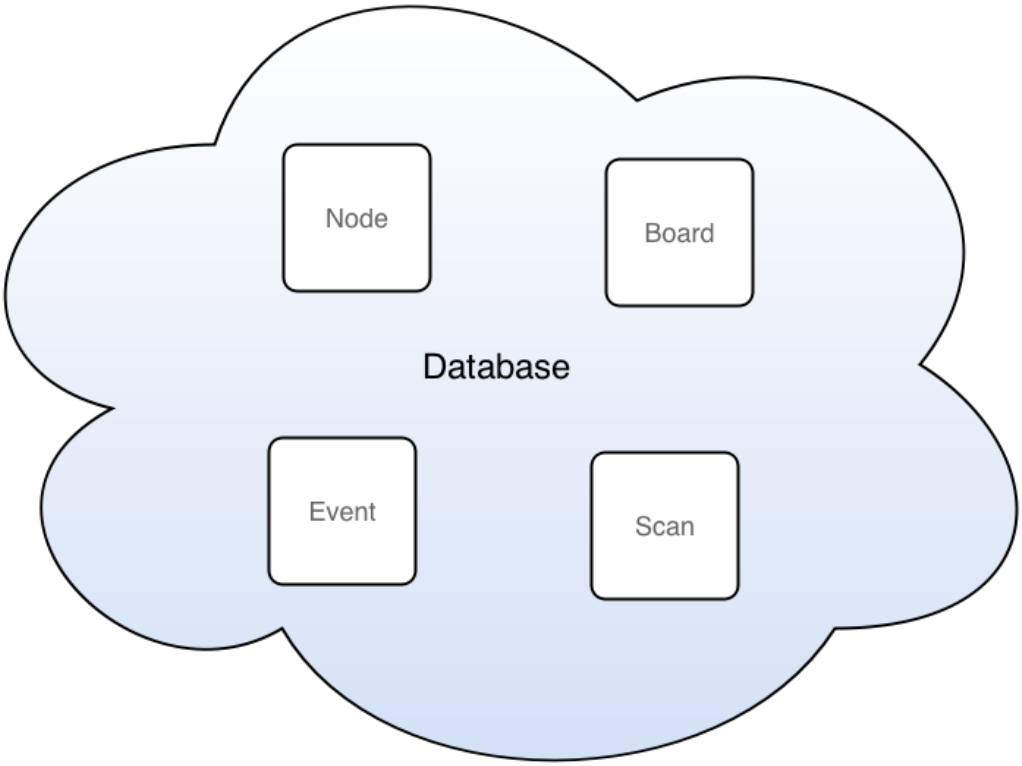

Figure 6. Overview of the database components.

Figure 6 above shows the four major components of the database. Indeed the structure of the data is not this simple. Each of the components consists of multiple database tables which are related to each other. The following list describes the data that the components include.

- The event component is at the heart of the database. It is the table that connects all other tables together. As the name suggests, it holds the information about events.

- The scan component holds the reference to the state of the board and reference to the node at the time of the scan.
- The board component holds the information about the actual board and its current and previous states.
- The node component holds the information about the node, node's current state and previous states.

The event component holds the necessary information about events. Events are the pieces of interesting information that are stored in the database. Events can be e.g. hardware errors and alarms. Every event is always associated with exactly one scan. In this way, the date when the event is stored in the database and the date when the event actually occurred will both be stored. The parser uses these timestamps to distinguish the events from each other.

The scan component holds necessary information about the scan. The scan's purpose is to make it possible to distinguish events, different hardware configurations and other necessary information related to a single scan. This makes it possible to see for example what hardware configuration the node had during the time when the event occurred.

The board component has all the information about the board's current and past state. This makes it possible to track down where a single board has been before when it was scanned by HAUDI. It also contains details about the current and past revisions, firmware versions etc.

The node component holds information about the node, its current software levels etc. It is very much like a board component.

5.3.3 Database performance

The techniques used to improve the performance of the database are covered in this chapter.

The database plays critical role in the performance so it must be designed carefully. Every single table in the database inherits a base table which has common columns

that every table in the database uses. In addition, the tables that have most entries will be inherited from the single master table. This means that there can be, for example 12 scan tables in the database. Every one of them is inherited from the master scan table. Data will then be inserted in these 12 child scan tables according to the month number, January being number 1 and so on. In this way, entries are distributed across 12 tables and this leads to smaller table sizes. This can greatly improve performance in the following situations.

- It will be more likely that indexes will fit in the memory.
- Some bulk operations are quicker and easier to perform against partition. Deleting old entries by month for example gets much quicker and easier. The partition holding the data can be just dropped.

The table inheritance and splitting described is called partitioning of the database and it is one of the most efficient techniques to fight against the tables that otherwise would grow unmanageably large [19] [20 p. 375].

Another key factor in database performance is indexing. Correctly implemented indexing can dramatically improve the query performance. What this indexing actually is? To put it simply, indexing means creating a helper structure in the database to make queries faster. The type of this structure is special, which makes lookup of a row by indexed column faster. Without index, the DBMS has to sequentially go through every row to be able to look up the queried one from the table which results linear lookup time. This is slow if the table contains a lot of rows and gets linearly slower when the size of the table increases. However with indexing, the lookup process is a lot quicker even if there are a lot of rows on the table. This is because there is no need for sequential lookup. Looking up the row with the index can be performed in time which grows logarithmically. This way the increase in a table size has little effect in query performance. However, there are downsides in having indexes. The database updates will get slower because the index must be updated every time when the table is updated. In addition, indexes will consume disk space and if they grow large enough not to fit in the memory, query performance will drop. The solution for this is the partitioning described earlier [20 p. 209] [21].

5.3.4 Detailed database structure

This section describes in more detail the tables in the database. Figure 7 below shows the database tables and how they are related to each other.

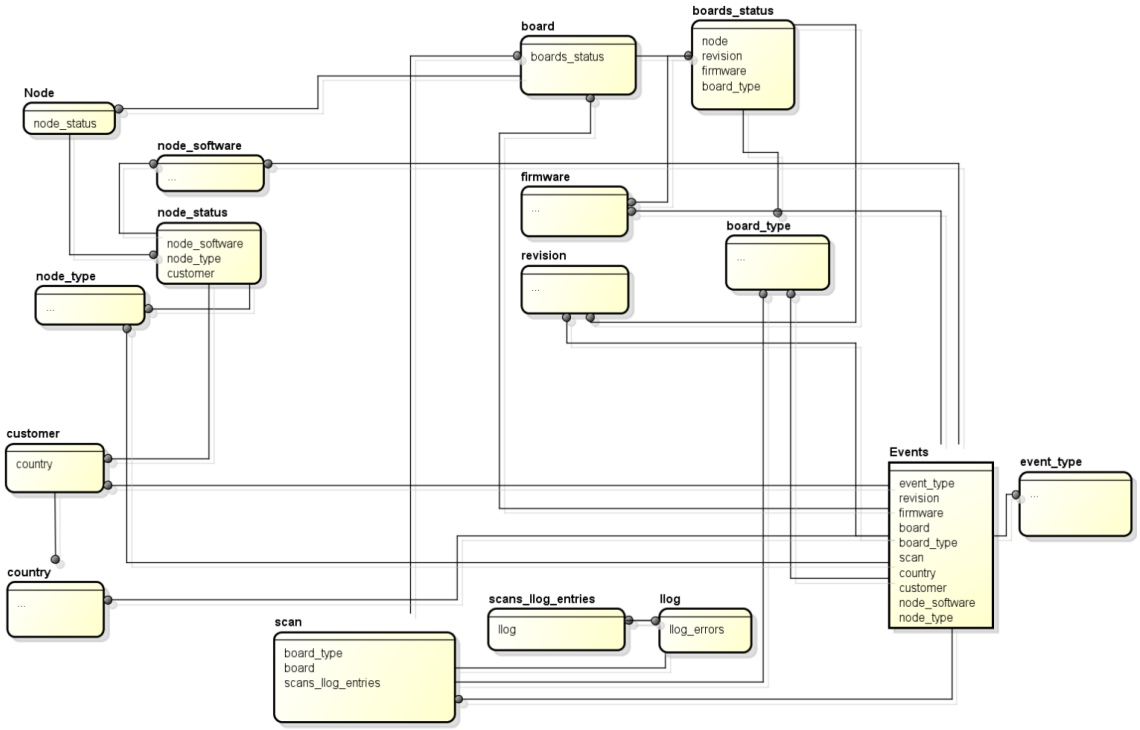

Figure 7. Detailed structure of the database.

As can be seen in Figure 7, the database structure is complex. Each of the boxes represent actual database table. The picture shows the relations of the database tables and only lists the names of the related tables as attributes. Any additional information is left out to keep the picture more readable. This additional left out information is, for example error codes, revision names, software levels and other necessary information. The eighteenth table which is not shown in Figure 7 is actually the table which all the other tables inherit. The table missing from the picture is called haudi_data and it consists of columns that are common to every table in the database. Most of the tables in the database make heavy use of indexing. Virtually all the columns that reference to other tables or are used in the most common queries are indexed.

5.4 Backend

This section covers the backend. First, the purpose of backend is described and then the database source, data source, logical structure and the RPC client are covered.

The backend is the part of the HAUDI which communicates with the database and frontend, and it is located between them. Backend can be thought of as a gatekeeper which guards the database access. It accepts requests to fetch, insert or update the database and satisfies these requests in a correct way. In this way database access is centralized and database integrity is easier to persist.

Backend is written in Perl which is the natural choice because it is also the language in which the ITK is written in. The backend offers an interface for the frontend to query the database and interface for the data access component to insert, query and update the database. In addition the backend is also responsible for log parsing. As Figure 4 showed the backend consists of 2 major components, namely the data access component and the RPC Client component. The name for the data access component comes from the fact that it is the only component which directly interacts with the collected data. The data access component is more complex compared to the RPC Client and it can be furthermore divided in three pieces. The following breakdown picture of the backend depicts the structure in more detail.

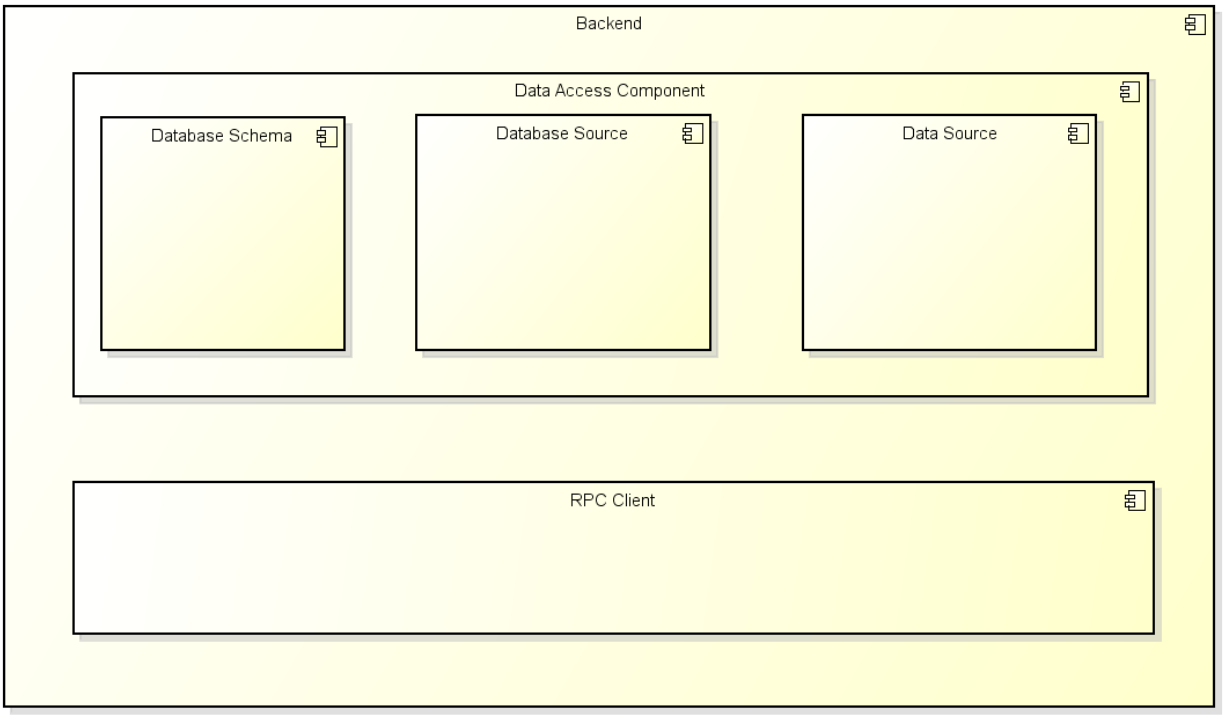

Figure 8. Structure of the backend.

As Figure 8 shows, the data access component consists of three modules. The two most important ones are the data source module and the database source module. The

third module is the database schema module which differs from the first two and it is only used when the database is set up. The modules are called data source, database source and database schema from now on.

5.4.1 Database source

Database source is the part of the program which directly interacts with the database, and it is the part of the data access component. All other parts of the program access database through the database source. For this purpose it offers an interface which the other parts of the program can use. The Database source is directly accessed by the data source but it is not directly accessed by the frontend. These topics will be covered more thoroughly in the data source and RPC client sections respectively.

Because Database source offers all the possible ways for HAUDI to interact with the database, it is complex. It consists of the functions for the data source component to add new data and check if data already exists in the database. It also has functions for the frontend to fetch data and search data with different parameters.

The reasons why the database access functionality is centralized are as follows:

- It provides a consistent way to access the database for all parts of the application
- It performs checks that queries have all the needed parameters
- It acts as a layer of security, offering only what is necessary, nothing more.
- Changes to queries are easy to make because they are defined only in one place.

The database source has 3 different set of functions. The first set includes all the necessary functions to create the database schema. This set of functions is only used to setup HAUDI and create an empty database structure including indexes, partitions and functions for partitioning. The second function set includes all the necessary functions for the HAUDI backend. These are mainly for the data source component. They include functions to insert, update and query data. The third function set consists of all the

functions for the frontend to query data. The functions for frontend have a minimal amount of data formatting and processing logic, because the idea is that the frontend does the processing which reduces the computational work of the backend.

5.4.2 Data source

The Data source component is the second part of the data access component. It is the log parser which provides all the data for the program, hence the name data source. The data source works in 2 phases.

The first phase loops through the log file provided for it. This phase does the actual file parsing and there are 5 steps, which we will look at next.

1. Extract the log files from the archive file.

2. The parser function is executed for each log file separately

3. The parser function splits the files in smaller pieces. The log is split by the commands which are used to collect the log.

4. The separate functions to parse the different command outputs are invoked

5. The parsing functions executed in previous step together insert the data into the logical structure

The steps from 2 to 5 are repeated until the parsing is done.

At this point the data is in the memory in a form of a logical structure, which will be covered in the next section. The logical structure may contain duplicate events, but there is no other unnecessary data. This is because the parsing functions invoked in step 4 make sure that only relevant data is collected. The parsing functions contain the regular expressions to match and extract the events from the log. Before the parsing functions insert the data in the structure, they call a hook function. Every parsing function has its own hook function and they do not insert the data in the logical structure before checking the return value of the hook function. Hook functions are made purely

for easing up the data manipulation before the data is inserted in the logical structure. They can be used, for instance, to discard events older than 1 month.

The second phase stores the data in the database, and phase 2 starts only after phase 1 is completed. At this point the data is in the memory hierarchically structured to make it easy to loop it through, and before any data is stored, there is a check if it is already in the database. All the steps in phase 2 which store data also perform the check if it already is in database, unless noted otherwise. The following steps are executed in phase 2.

1. The node and its state are stored in the database and if it was in the database already, there is check if the state of the node has changed.

2. Associate the board with the correct node and store the board and its status to the database.

3. New the scan entry is created. There will always be a new scan for every board, no checks performed in this step. The scan will be associated with the board from the previous step.

4. Different pieces of information, restart information for instance, is associated with the board and stored.

5. The events are looped through, associated with the correct board and stored.

The steps from 1 to 5 are repeated until there is no data left. This phase is the most computational intense in the process simply because there is a significant amount of database queries executed in a loop.

After these two phases described above, the data about nodes, boards and events are stored in database.

5.4.3 Database schema

The database schema has the logical structure of the database stored in it. This means that it has definitions of every single database table in it, including column names, data

types and indexed columns. It also has all the functions needed for the partitioning which include partition trigger functions for database inserts and dynamic partition table creation functions. The database schema also has helper functions to extract information about the database tables. These helper functions are used by data source and database source when setting up the database.

5.4.4 Logical structure

The logical structure is in the memory structure which the data source builds from the logs. It consists of the node, boards, events and other additional information. The logical structure makes heavy use of Perl`s hashes. Hashes are basically key-value pairs stored in the memory and they are used because the keys which are used to find values are unique. This uniqueness means that there cannot be two duplicate keys in a hash, and this makes hashes perfect for creating tree like structures. In addition hashes in Perl do not slow down even if the size of the hash grows large. This is mainly because the internal implementation of hashes is extremely efficient in Perl [22 p. 107]. These features make it possible to build hierarchical structures efficiently and easily. Figure 9 depicts a simplified version of the logical structure.

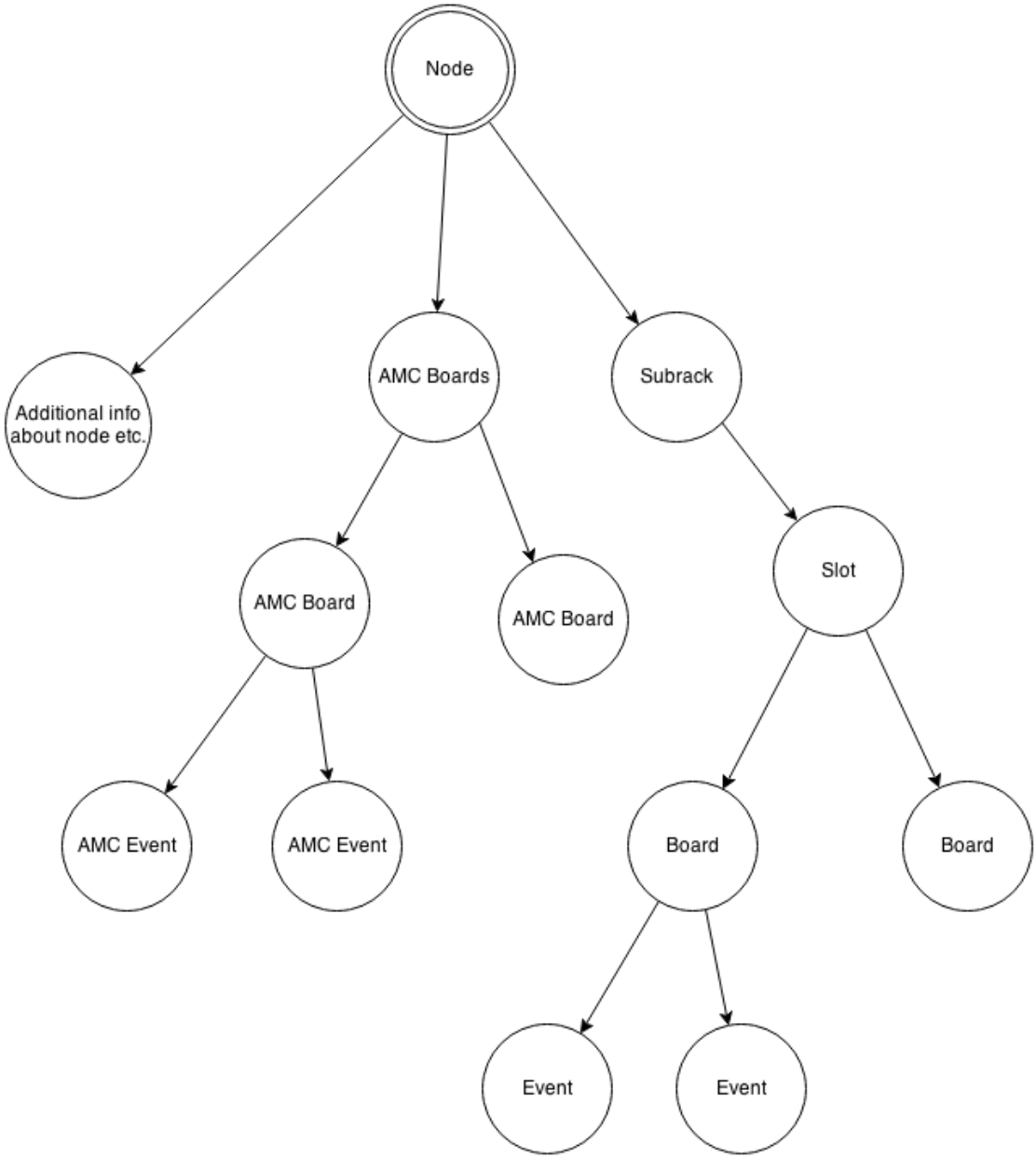

Figure 9. Logical structure.

As it can be seen in Figure 9 above, the structure is hierarchical and tree like. Node name is the key, a root of the tree which is used to find the data associated with the node. For instance, under the node there is subrack and under subrack, there is a slot and under the slot the board itself. The events and the information associated with them are as a list under the board in the structure. This keeps them organized and the event is always associated with the correct board and the board is always associated with the correct slot and so on. By traversing through this tree structure it is easy to find necessary data and data remains well organized. The tree structure is easy to traverse

through and it brings some benefits over a flat list like structure. For instance the events do not need to have extra information stored with them about the board which the event is associated with; the information is built in the structure. Another benefit is the easy manipulation of data, for instance it is easy to move or copy a whole branch of data. Moreover if the board and all events associated with it must be deleted, in a tree like structure it's easy just to delete the board. This will automatically delete everything that is under the board in the structure as well.

5.4.5 RPC Client

The RPC client is the interface for the frontend to fetch data from the database by backend. It consists of the functions which can be used to fetch data from the database utilizing the database source. The RPC client only allows reading data from the database, it does not offer a data update or data insert functionality for the front end. The database source has a set of functions meant to be used only by the RPC client.

RPC stands for Remote Procedure Call and it is a form of inter-process communication. Inter-process communication makes it possible for a program to call another program in a different server for example. The RPC client uses JSON RPC because ITK already utilizes it, and this makes it a natural choice. JSON RPC is a lightweight and stateless RPC protocol which is designed to be simple. It has strict rules about how data must be sent and how the receiver of the data must reply [23].

The JSON in the JSON RPC stands for JavaScript Object Notation. This is a name of the format in which the data is transferred. JSON is a human readable simple data format which consists of attribute-value pairs. JSON can be used in frontend without doing any data conversion [24]. This is because the frontend is written in JavaScript and it can handle JSON without any processing or formatting.

The functions which the RPC client consists of are simply for data fetching. Some of the functions need parameters and they will result in an error message response if parameters are missing or of the wrong type. The functions will return either error code or the result like JSON RPC specification defines.

5.5 Frontend

The frontend is written in Javascript using AngularJS framework. This chapter covers AngularJS first and then the structure of HAUDI's frontend is covered in more detail.

5.5.1 AngularJS

AngularJS is an open-source framework that is maintained by Google. AngularJS is created to assist for writing single-page applications and it offers model-view-controller capability. AngularJS is a powerful framework and to put it simply, it simply extends HTML by adding tags in it, which are called directives. When AngularJS detects one of these directives it does what it is programmed to do. In the model-view-controller model these new directives live in the view part, which is a normal HTML file containing these special directives that AngularJS will detect and understand. The model part consists of a data processing functionality and the controller part handles the routing of requests that the user issues. AngularJS was chosen after a careful review of different technologies and it was chosen because it's robust and yet simple and a well compatible framework. It includes everything necessary to create vivid web applications using thick client architecture. In addition it has thorough documentation available [25].

As an example, an ng-model tag dynamically binds the data model to an html input field. The following html snippet illustrates how it is used.

```
<input ng-model="my_value" />
<br />
{{ my_value }}
```

This is how it works: when the user inputs something to the field, it will be immediately updated to the variable called my_value which will be printed on the web page immediately. Here, the ng-model is the directive and {{ my_value }} binds the variable to the html, prints it and dynamically updates when the data which the variable contains changes. All this happens without the need to refresh the web page.

5.5.2 Overview

The frontend is the part of the software which the user sees and interacts with. It is responsible for fetching data from backend, processing it and rendering the data visualizations. Frontend consists of 2 major components which are views & routing and services. In addition to the previous there is also a component used in data visualizations and it is covered in the data visualization section later in this chapter. Figure 10 below depicts the structure of the frontend.

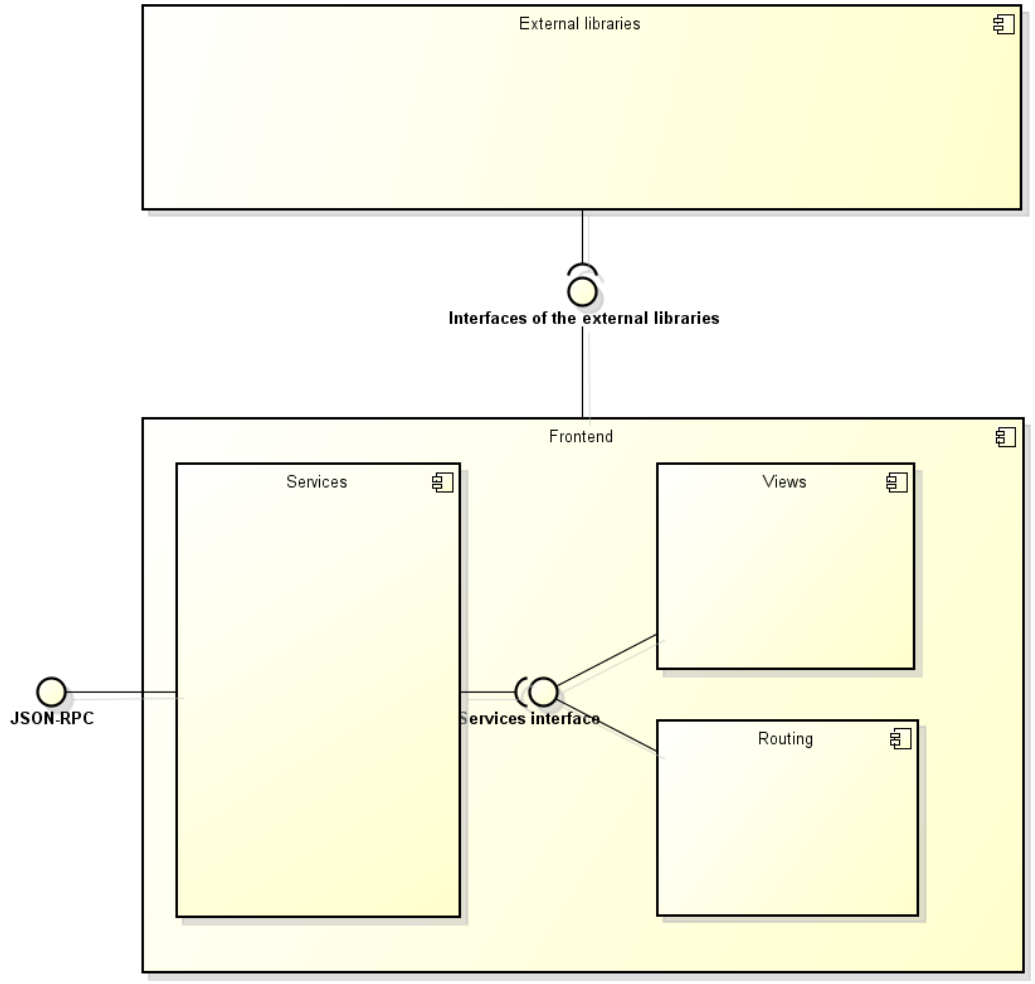

Figure 10. Structure of the frontend.

The different components can be clearly seen in Figure 10. The services component handles the connection to the backend and the data processing as well. The services component is the most complex part of the frontend. The routing component handles the loading of the correct page and resources needed before the user enters the URL. The views component consists of the HTML markup for the web pages along with the

tags which AngularJS interprets. The external libraries component is a collection of the necessary libraries used to create the visual appearance of the web pages and those will be covered in more detail in data visualization section.

5.5.3 Views and routing

The routing part of the frontend is called router. Router is set of rules about how the frontend should respond when the user enters an URL. It uses white listing to list URL s that are allowed. In other words, this means that there is a list of URLs that are allowed and everything that does not fall in this category is discarded and the user is redirected safely to the front page of the web application.

Router also handles the pre-loading of resources. Basically this means that the router blocks the user until the pre-loading is done and only after this the web page is rendered. This is necessary when the web page in question needs initial data from the database. The waiting occurs in this case because the data fetching from the RPC client is asynchronous. The web application must wait for the backend to respond before it is allowed to continue to load the web page which needs initial data. Otherwise there will be reference to a data which is not loaded yet, which causes the application to fail by referencing data which does not exist.

Views are the part of the frontend that will be shown to the user. They are HTML markup files with additional AngularJS tags which are called directives. There are several different directives that come with AngularJS and the programmer can add their own directives as well. Directives can do a simple job e.g. format a date or reverse text or then they can do complex things like draw a chart out of data.

There is also a part called scopes which belongs together with services and router, and in fact the services and router are built on top of scopes. The scope glues the application together and isolates the different parts of the application. Scopes handle the initial formatting of the variables and hold them in their own isolated namespace. This helps to create bigger applications with a lot of data and variables by reducing memory usage of the application and by offering namespaces. In other words scopes divide the applications clearly in smaller easily manageable pieces.

5.5.4 Data visualization

Data visualization uses SVG to visualize the data. SVG is an abbreviation of Scalable Vector Graphics. SVG is a set of technologies built in modern web browsers which make it possible to draw vector graphics on web sites. Vector graphics has a great advantage over the traditional way of using pre rendered pictures and it is the scalability. Scalability means that no matter in which size the website is viewed, the picture and its proportions remain the same. This makes it possible to view the picture from a laptop, from a big screen or even from a tablet or cell phone, the information contained in the picture remains undistorted.

The SVG is implemented by all major modern web browsers and while SVG can be used and accessed relatively easily by using JavaScript, there are libraries for especially drawing charts with SVG. These libraries implement the basic features needed to create the charts e.g. scales and domains. These are purely made to speed up the development process of charts by offering already tested and functioning modules to build up the charts. In addition the developer who uses the library does not need to have a thorough understanding of internals of the charts and the math behind them, e.g. a programmer does not have to worry about how logarithmic scales are converted to linear scales.

The library used in HAUDI is called Data Driven Documents (D3 from now on). D3 is a JavaScript library for manipulating documents based on data [26]. The bar chart in Figure 5 is produced by D3 library.

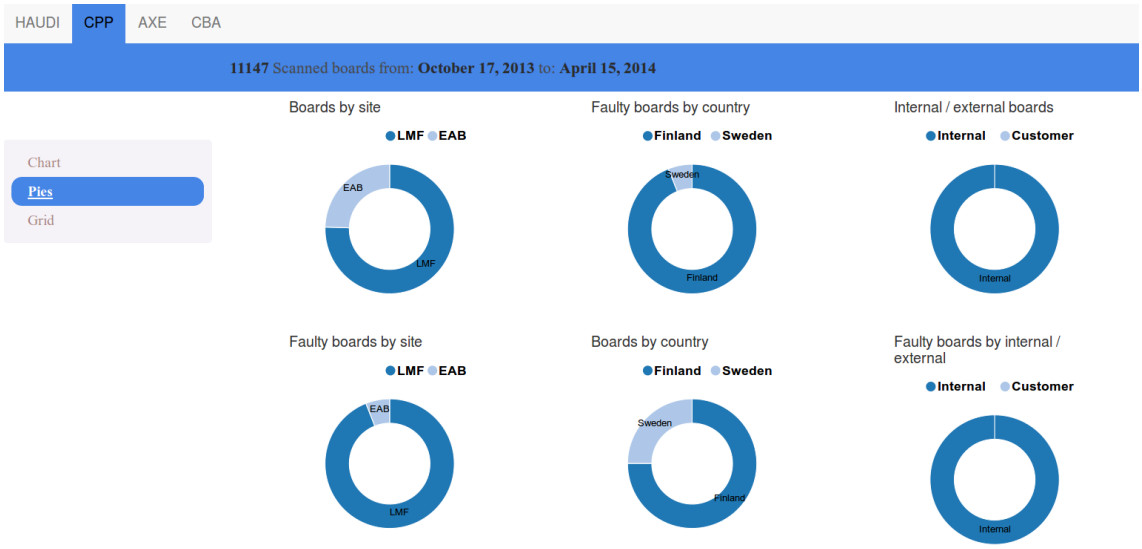

Figure 11. Pie charts created by D3.

D3 is written in JavaScript as is everything that interacts with SVG. It was chosen because of its open source library and it also has an active community and it has good documentation. In addition, there are lots of readymade charts for it published by the developers and those can be easily modified and further developed to satisfy the needs of HAUDI.

5.5.5 Services

The services component has the business logic in it and it contains the following types of functions.

- Functions that fetch data from backend.
- Functions that process the fetched data
- Functions that help the frontend to work and respond to user input.

The data fetching functions are simple. They check and handle the possible parameters and format the data to the format that JSON-RPC requires. After this the request is sent and AngularJS handles the response by utilizing promises. Promise is just another

name for waiting the server to respond and telling for the application when the response came and the data is ready to be used.

Data processing functions consist of functions to format data for the charts, tables and other elements which require data from the backend. This is a necessary step because the data from the backend is not processed. The reason for this comes from the thick client architecture; it's better to handle the data formatting in the frontend so the backend has less load and can support more users simultaneously.

The helper functions consist of functions that do some specific task for the frontend. An example of this is for instance a function which awakens the necessary data fetching and processing functions when the user clicks a button or fills a text field on the web page.

The services component has most of the functionality built in it, and other components just ask the services component to do the work for them.

5.6 Summary

This chapter dealt with the design and implementation of HAUDI.

Usability was the first topic and from usability it's good to remember that even if the software is well made and advanced, the users won't use it if usability is bad. In addition usability is the single most important thing when it comes to the amount of visits and user loyalty of web sites.

The second topic covered was database related topics. From those it's good to remember the following things.

- Database structure is important in terms of overall performance
- Database is designed to be as scalable as possible
- Performance of the database is improved by indexing and partitioning the tables

- Database structure of HAUDI is complex

- Database is the single most critical component in view of performance

6 Results

This chapter deals with the results of thesis.

HAUDI offers a view inside quality related data no one has seen before. It offers time series analysis among other things and this reveals totally new information which was unknown before. This is because it would be too difficult to gather all the data manually and process it correctly. As a result HAUDI will hopefully end up to be used as a part of a process to improve the hardware quality and act as a source of hardware quality information including the present state of quality and the past state of quality.

HAUDI was built to increase co-operation and information passing between organizations. Only time will show whether this will happen, but there are already encouraging examples that it will happen. Another objective was to make a clear distinction between hardware and non-hardware related quality issues. HAUDI does this by default, because it only scans and stores hardware related issues.

HAUDI was successfully built to work with ITK and it makes heave use of its services. This was one of the requirements for HAUDI and it was met, although it’s not yet released with ITK distribution. This is important for the future integration and release process of HAUDI.

At the moment HAUDI is used to demonstrate its full potential and the final integration work with ITK is about to begin. There has been interest towards HAUDI and this interest has produced requests to add new features in it.

7 Further development

Because HAUDI is a proof of concept, there is still a great deal of work to do with it. The next big step should be the final integration and release of HAUDI with ITK. In addition there is interest to get HAUDI to operate on other platforms than CPP also. This will greatly expand the amount of nodes HAUDI can collect the data from and new data collection and parsing methods have to be implemented. Moreover there is a lot of work to do with optimizing the performance of the database and the parser. The database structure could use some improvements, generalizations and partitioning should be implemented in a more efficient way. Moreover the parser should be rewritten to do data comparison completely in memory. This would dramatically drop the amount of database queries and thus speed up the parser. Moreover there is a need for a way to report which data the parser actually did parse and find from the log files. In this way it would be a lot easier to verify that the parser found all the necessary data from the log files.

8 Summary

The purpose of this thesis was to create a proof of concept tool for real-time quality monitoring tool called HAUDI. The need to create this tool came from the lack of hardware specific quality monitoring tool which allows statistical real-time analysis of quality data. The purpose of this tool is to show the current state of hardware quality for everyone who might be interested in it. This on the other hand would help organizations to co-operate better and bring forward the issues there are found in hardware quality. The goal of this is to drive individuals to take action and resolve the cause of the issue.

At the beginning of this thesis quality related topics were handled. After this HAUDI's purpose, goals, requirements and limitations were discussed. The limitations section included a description of the network, CPP and M-MGw. Next the design and implementation details of HAUDI were handled including database structure, frontend, backend and usability.

HAUDI was able to reveal new information about the hardware from the hardware of the test plant. It brought the information available in an easy way and for the first time, it was possible to easily see how the amount of new hardware related issues was related for example with a software level. As a result, HAUDI was proven useful and it awakened interest in different organizations. Integrating HAUDI with ITK and productizing it is clearly a justified next step.

References

1 Larry Webber, Michael Wallace 2012, Quality Control for Dummies Wiley Publishing Inc.

2 Pete Pande, Larry Hollp 2001, What Is Six Sigma? McGraw-Hill.

3 The Syslog Protocol [web document, accessed 22.3.2014] Available: http://tools.ietf.org/html/rfc5424

4 Nagios Overview [web document, accessed 22.3.2014] Avalilable: http://www.nagios.org/about/overview

5 Munin [web document, accessed 22.3.2014] Available: http://munin-monitoring.org/

6 Fault detection and isolation [web document, accessed 22.3.2014] Available: http://en.wikipedia.org/wiki/Fault_detection_and_isolation

7 Control theory [web document, accessed 22.3.2014] Available: http://en.wikipedia.org/wiki/Control_Theory

8 Powers of 10: Time Scales in User Experience [web document, accessed 19.3.2014] Available: http://www.nngroup.com/articles/powers-of-10-time-scales-in-ux/

9 Tool for network level configuration and auditing in mobile backbone network, master's thesis. Espoo 2011.

10 CPP System Description document, Ericsson.

11 Ericsson Media Gateway for Mobile Networks, Ericsson.

12 Fat Client [web document, accessed 6.12.2013] Available: http://en.wikipedia.org/wiki/Fat_client

13 Predicting Users' First Impressions of Website Aesthetics With a Quantification of Perceived Visual Complexity and Colorfulness, Reinecke, Yeh, Miratrix, Mardiko, Zhao. Available: http://www.eecs.harvard.edu/~kgajos/papers/2013/reinecke13aesthetics.pdf

14 Response Times: The 3 Important Limits [web document, accessed 6.1.2014] Available: http://www.nngroup.com/articles/response-times-3-important-limits/

15 Steve Krug, 2006. Don't Make Me Think: A Common Sense Approach to Web Usability, 2nd Edition. New Riders.

16 Object-relational database [web document, accessed 17.3.2014] Available: http://en.wikipedia.org/wiki/Object-relational_database

17 What is PostgreSQL? PostgreSQL documentation v. 9.3 [web document, accessed 23.11.2013] Available: http://www.postgresql.org/docs/9.3/interactive/intro-whatis.html

18 About PosgreSQL [web document, accessed 29.11.2013] Available: http://www.postgresql.org/about/

19 Partitioning. PostgreSQL documentation v. 9.1 [web document, accessed 30.11.2013] Available: http://www.postgresql.org/docs/9.1/static/ddl-partitioning.html

20 Gregory Smith 2010, PostgreSQL 9.0 High Performance.

21 Database Index [web document, accessed 23.11.2013] Available: http://en.wikipedia.org/wiki/Database_index

22 Randal L. Schwartz 2011, Learning Perl 6th edition. O'Reilly Media.

23 JSON-RCP 2.0 Specification [web document, accessed 20.2.2014] Available: http://www.simple-is-better.org/json-rpc/jsonrpc20.html

24 RFC 4627 [web document, accessed 26.2.2014] Available: http://tools.ietf.org/html/rfc4627

25 AngularJS documentation [web document, accessed 2.3.2014] Available: http://docs.angularjs.org/api

26 Data Driven Documents documentation [web document, accessed 1.3.2014] Available: https://github.com/mbostock/d3/wiki